Guest Editors' Introduction: Agile Software Development: It's About Feedback and Change

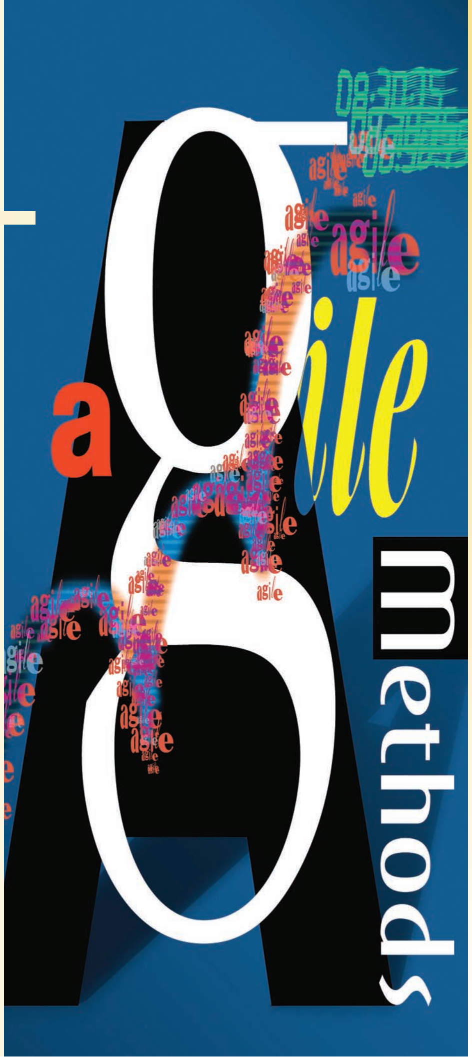
Article *in* Computer · July 2003

DOI: 10.1109/MC.2003.1204373 · Source: IEEE Xplore

Alistair Cockburn
Humans and Technology, inc.

Agile Software Development: It's about Feedback and Change

Currently, the focus is on determining how to blend agile methodologies with plan-driven approaches to software development.



Laurie Williams
North Carolina State University

Alistair Cockburn
Humans and Technology

Agile software development has hit a nerve in the software development community. Some people argue vociferously for it, others argue equally against it, and others are working to mix agile and plan-driven approaches. Many more people wonder just what agility is.

WHAT IS AGILITY?

In many development methods, particularly plan-driven ones, work begins with soliciting and documenting a complete set of requirements. Beginning in the mid-1990s, many found this initial requirements documentation step frustrating and, perhaps, impossible.[1] Both technology and the business environment kept shifting during the project, and both the requirements and project plans got out of date within even relatively short projects. Customers became increasingly unable to definitively state their needs up front.

As a result, practitioners developed methodologies and practices to embrace, rather than reject, higher rates of change. These methodologies were developed on three different continents: the Dynamic Systems Development Method in Europe; Feature-Driven Development in Australia; and Extreme Programming, Crystal, Adaptive Software Development, and Scrum in the US. Although their practices and philosophies shared fundamental similarities, these practitioners authored their methodologies independently.

Underlying values

In February 2001, 17 of these practitioners and authors met in Snowbird, Utah, to discuss the fundamental similarities of their experiences and their then-called *lightweight* methodologies.[2] Finding that their work habits had much in common, they recognized that the lightness in their work process provided a means to achieve the more relevant end: customer satisfaction and high quality. They categorized their methodologies as "agile"—a term with a decade of use in flexible manufacturing practices.

The participants wrote the "Manifesto for Agile Software Development" (www.agilemanifesto.org), which describes the four comparative values underlying the agile position:

- individuals and interactions over processes and tools,
- working software over comprehensive documentation,
- customer collaboration over contract negotiation, and
- responding to change over following a plan.

These four comparative values recognize the importance of the items on the right of each comparison, A point of commonality for all agile methods is the recognition of software development as an empirical process.

but assert the dominance of the ones on the left.

Empirical versus defined processes

Another point of commonality for all agile methods is the recognition of software development as an empirical (or nonlinear) process. In engineering, processes are classified as defined or empirical.[3] A defined process is one that can be started and allowed to run to completion, producing the same results every time.[4] Assembling an automobile is such a process. Engineers can design a process to assemble the car and specify an assembly order and actions on the part of the assembly-line workers, machines, and robots. Generally speaking, if the manufacturing process follows these predefined steps, it will produce a high-quality car.

Software development cannot be considered a defined process because too much change occurs during the time that the team is developing the product. It is highly unlikely that any set of predefined steps will lead to a desirable, predictable outcome because requirements change, technology changes, people are added and taken off the team, and so on.

In an engineering context, empirical processes necessitate short "inspect-and-adapt" cycles and frequent, short feedback loops.[4] These short inspect-and-adapt cycles can help agile methodologies better handle the software industry's conflicting and unpredictable demands.

Other empirical methods that are in synch with the Agile Manifesto can also be considered agile. Agility, would-be agility, and various competitive counterparts have become hot topics in conferences and magazines—including IEEE Computer Society publications.[5-8] There are now both books and annual international conferences on the topic.

CURRENT CONVERSATIONS

As practitioners have gained experience with the use of agile methods, the focus of their conversations about this approach to software development has changed.

Suitable circumstances

In 2001, the conversation around agile was: What is it really? Is it old, new, a fad, crazy, or what?

We found that agile development is not particularly new. Software developers have been sporadically using the techniques since at least the 1960s. What is new, if anything, is the bundling of the techniques into a theoretical and practical framework and the strong, sometimes vehement, declaration of their importance.

In 2001, enough sample projects had been collected to make the case that agile development works under "suitable circumstances." These circumstances generally can be categorized as non-safety-critical projects with volatile requirements, built by relatively small and skilled colocated teams.

Scalability

In 2002, the agile conversation focused on two questions: How do we make it scale beyond these "suitable circumstances?" Does agility contradict the Software Engineering Institute's Capability Maturity Model and ISO 9000 rules?

A few projects of several hundred people have used the agile framework succesfully.[9] However, during 2002, agile practitioners and other researchers concluded that the agile value set and practices best suit colocated teams of about 50 people or fewer who have easy access to user and business experts and are developing projects that are not life-critical. Although the agile value set might be adopted (and may be adapted) under other circumstances, the boundary conditions for truly agile behavior seem to be fairly set.

A close but liberal reading of the CMM reveals little that would prevent CMM-concerned teams from using agile methods.[10] However, some common CMM practices are at odds with the agile value set because of their reliance on processes, plans, and delivering to original contract. Generally, people willing to spend the money on CMM certification are less interested in the agile value proposition, while those needing agility for business reasons are less interested in getting CMM or ISO 9000 certification.

Adaptability

In 2003, the conversation has changed again: Can we blend selected agile practices with our "regular" practices? How much change is necessary when transitioning to and using agile methods? How can agile practices improve the quality of our products?

Development groups are trying pair programming without any other agile practices, test-first development without pair programming, and short delivery cycles without pair programming or test-first development, and trying to determine how to get any of these practices to work with internationally distributed teams. Researchers worldwide are gathering these agile subset experiences and designing and staging additional experiments to empirically assess the efficacy of isolated practices or agile subsets.

We see agile philosophies and practices dispersed over a wider range of people, projects, and organizations. For example, many CMM and ISO 9000 organizations now think that partial adoption of agile practices, when handled with care, might increase their efficiencies without damaging their certifications.

Making changes

As agile development spreads from the early adopters to a larger population of developers and development managers, the question of change is coming to the fore.

Agile development carries with it the implication of changing work habits. For those who have not used agile techniques, the change involves trying out and selecting appropriate ones. For those already using some of the techniques, the change involves adjusting and improving their use over time. Some proportion of the development community has no interest in change, however, which may impose an inevitable limit to the penetration of agile techniques.

Agile development can affect the power structure within an organization because it spreads out the decision-making authority. Executives are expected to make and take accountability for business-level decisions, deferring to the developers on technical issues such as development techniques and time estimates. The developers are then accountable for these issues.

This decision-making approach differs from what we see in many organizations. In some, the programmers make many key decisions, including those pertaining to business, process, and systems requirements, while their managers either happily or reluctantly go along for the ride. In other organizations, programmers have limited decision-making authority, and they are treated as little more than clerical workers.

In either case, the move to agile development, with its reallocation of decision-making authority and accountability, can cause consternation. We encounter managers who report that they consider agile development to be a license for developers to hack, but we also encounter developers who contend that agility is merely a ploy to let managers micromanage their work. Just how to handle this power shift remains an open question.

Finally, some question the ability of agile practices to improve product quality. Agile practices stress early involvement of test groups and the need for rapid feedback, but they don't say much about the relationship between the development and quality assurance groups. Integrating quality assurance into projects to enhance their agility requires making changes in the relationship between these groups and even in hiring practices.

Integrating quality assurance into projects to enhance their agility requires making changes.

IN THIS ISSUE

The articles in this issue capture some results from earlier conversations and set the stage for new ones.

In their short and colorful "Agility through Discipline: A Debate," Kent Beck and Barry Boehm suggest that agile and plan-driven development exist in different ecological niches and that each will thrive or dominate depending on the "current weather conditions." Somewhere between the elephant, the monkey, the mouse, and the dinosaur, their debate conveys the message that both approaches require discipline to work—although with some different shadings of what discipline means—and that each works well in specific circumstances.

Craig Larman and Vic Basili's article, "Iterative and Incremental Development: A Brief History," traces this cornerstone agile practice back to the 1950s and even before. They cite NASA's 1957 Mercury project, which used half-day increments, and a 1969 report to IBM management that recognized "the futility of separating design, evaluation, and documentation processes in software-system design." Their article includes a quote from Gerald Weinberg, who writes of an experience at about this time "where the technique used was, as far as I can tell, indistinguishable from XP."

Barry Boehm and Rich Turner examine the agile versus plan-driven debate and provide recommendations for how and when to mix the two approaches. In "Using Risk to Balance Agile and Plan-Driven Methods," they highlight both the home grounds and risks associated with each approach. They describe how they would choose which approach to use and under what circumstances they would mix them, highlighting their ideas with three sample projects that should use different combinations of agile and plan-driven practices. They characterize the projects using a five-point radial diagram that isolates important project features driving this decision. The diagram also captures some part of the organization's ability to undertake change.

Boehm and Turner list three categories of risk that can affect the choice of how heavily to bias a

Working in an environment in which all projects must use a common process adds a new wrinkle to introducing agile processes.

project toward the agile or plan-driven side:

- risks stemming from an agile approach: scalability, criticality, design simplicity, staff churn, and staff skills;
- risks stemming from a plan-driven approach: emerging requirements, constant change, need for rapid results, and staff skills; and
- general environmental risks: technology uncertainties, diverse stakeholders, and complex systems.

In listing these risks, they establish a framework for examining the project experiences and characteristics of various development approaches, a framework echoed in the other three articles.

In "Developing Complex Projects Using XP with Extensions," Martin Lippert and his coauthors describe how they have dealt with circumstances encountered on projects using (mostly) Extreme Programming. There is an echo of the Boehm-Turner list in this work, even though the authors arrived at their cyclical approach to software development independently, based on specific project experiences.

The authors discuss how they handled diverse and conflicting interest groups driving a project, lack of testing on the customer's side of the equation, complex systems and imprecise requirements, long-term planning, dependencies among nonprogramming tasks, and the need for requirements definition at least for funding purposes. It is refreshing that they identify the special skills the people in key roles need, including the product manager, the author-critic, and even the customer. Their experiences provide additional practices for agile developers to consider using and demonstrate the benefits of mixing selected traditional approaches with standard agile and situationally derived practices.

In "Introducing an Agile Process to an Organization," Mike Cohn and Doris Ford describe their experiences with introducing Scrum into seven organizations over a four-year period and reflect on what they learned along the way. They describe watching some developers add formal documents back into their work practices—"we invariably found programmers who enjoy producing noncode artifacts far more than they are willing to admit"—and recall other overzealous developers who view agile development as meaning they must not think ahead.

In light of the Beck-Boehm debate around discipline, Cohn and Ford provide an interesting description: "They did not have the discipline XP requires and, while paying lip service to XP, they were actually doing nothing more than hacking." These authors discuss their experiences with some Boehm-Turner issues such as distributed development and describe other issues that popped up on their projects, such as encountering developer resistance to lightening the process, working with the human resources department, using testers, and managing the project and its interfaces to other groups.

Finally, in "Migrating Agile Methods to Standardized Development Practice," Mark Lycett and his coauthors address a new wrinkle to introducing agile processes into an organization: working in an ISO 9000 or CMMI environment or one in which a large corporation has mandated that all projects must use a common process. They make clear the need for a company to establish quality of product, process, and service while still letting different projects work in the various ways that suit them best.

Reporting on their experiences with a large company, these authors describe their particular approach to project-specific process tailoring: a set of process "patterns" that can be mapped to the CMMI framework. For ISO 9000, they adopted a pattern-based framework "to supplant repeatability with consistency, while providing the audit trail necessary for assessment." For good project-specific results, they revisited their use of the patterns *after each iteration* and found that "making the decision process visible is extremely valuable because it forces management to actively determine whether an activity or artifact is both necessary and sufficient." Echoing and extending the first Agile Manifesto value, they describe their framework as "suggesting suitable techniques for collaboration, interaction, and communication."

Lycett and his coauthors offer their own resolution of the agile versus engineering debate: "Agile process does not fly in the face of engineering practice. If used thoughtfully, it provides a clear mandate for making engineering practice lean and well focused."

It is a pleasure to introduce these articles in this special issue. They capture the state of the current conversation, and we hope they serve as the catalyst for future ones. ■

References

1. J. Highsmith, *Agile Software Development Ecosystems*, Addison-Wesley, 2002.

2. M. Fowler and J. Highsmith, "The Agile Manifesto," *Software Development*, Aug. 2001, pp. 28-32.
3. B.A. Ogunnaike and W.H. Ray, *Process Dynamics, Modeling, and Control*, Oxford Univ. Press, 1994.
4. K. Schwaber and M. Beedle, *Agile Software Development with SCRUM*, Prentice Hall, 2002.
5. B. Boehm, "Get Ready for Agile Methods, with Care," *Computer*, Jan. 2002, pp. 64-69.
6. A. Cockburn and J. Highsmith, "Agile Software Development: The People Factor," *Computer*, Nov. 2001, pp. 131-133.
7. J. Highsmith and A. Cockburn, "Agile Software Development: The Business of Innovation," *Computer*, Sept. 2001, pp. 120-122.
8. L. Williams, "The XP Programmer: The Few-Minutes Programmer" (guest editor's introduction), *IEEE Software*, May/June 2003, pp. 16-20.
9. R. Crocker, *Large-Scale Agile Software Development*, Addison-Wesley, 2003.
10. M. Paulk, "Extreme Programming from a CMM Perspective," *IEEE Software*, Nov./Dec., 2001, pp. 19-26.

***Laurie Williams** is an assistant professor of computer science at North Carolina State University. Her research interests include agile software development methodologies and practices, software reliability, software testing, and software engineering for secure application development. She received a PhD in computer science from the University of Utah. Williams is the coauthor of* Pair Programming Illuminated *and* Extreme Programming Perspectives *(both Addison-Wesley, 2003). She is a member of the IEEE and the ACM. Contact her at williams@csc.ncsu.edu.*

***Alistair Cockburn**, a coauthor of the "Manifesto for Agile Software Development," received a PhD from the University of Oslo on the subject of people and methodologies in software development. He has published widely on formal specification, object-oriented design, and development processes, but is most widely recognized as the author of* Writing Effective Use Cases *and* Agile Software Development. *Many of his publications are available online at alistair.cockburn.us. Contact him at acockburn@aol.com.*